

LAPORAN PRAKTIKUM STRUKTUR DATA
QUEUE PADA JAVA



DOSEN PENGAMPU :

Dr. Wahyudi, M.T.

OLEH :

RIFKI YULIANDRA

NIM.2311532011

DEPARTEMEN INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
UNIVERSITAS ANDALAS
PADANG
2024

Queue Pada Java

I. TUJUAN

- 1.1. Memahami apa yang dimaksud dengan queue
- 1.2. Memahami cara membuat queue
- 1.3. Memahami cara membuat stack dengan menggunakan class input
- 1.4. Memahami cara mereverse data queue.

II. TEORI

Dalam dunia pemrograman, khususnya Java, **Queue** (antrian) merupakan struktur data fundamental yang mengikuti prinsip First-In-First-Out (FIFO). Bayangkan antrian di toko, orang yang pertama masuk antrian adalah orang yang pertama dilayani. Konsep ini berlaku pada queue, elemen yang pertama kali ditambahkan ke dalam queue akan menjadi elemen pertama yang diambil dan diproses.

Queue termasuk ke dalam kerangka kerja koleksi Java (Java Collection Framework) yang menyediakan antarmuka (interface) `Queue`. Interface ini mendefinisikan operasi dasar yang dapat dilakukan pada antrian, seperti menambahkan elemen (`enqueue`) dan menghapus elemen (`dequeue`). Selain itu, terdapat operasi lain untuk memeriksa apakah antrian kosong (`isEmpty`) dan mengetahui jumlah elemen di dalam antrian (`size`).

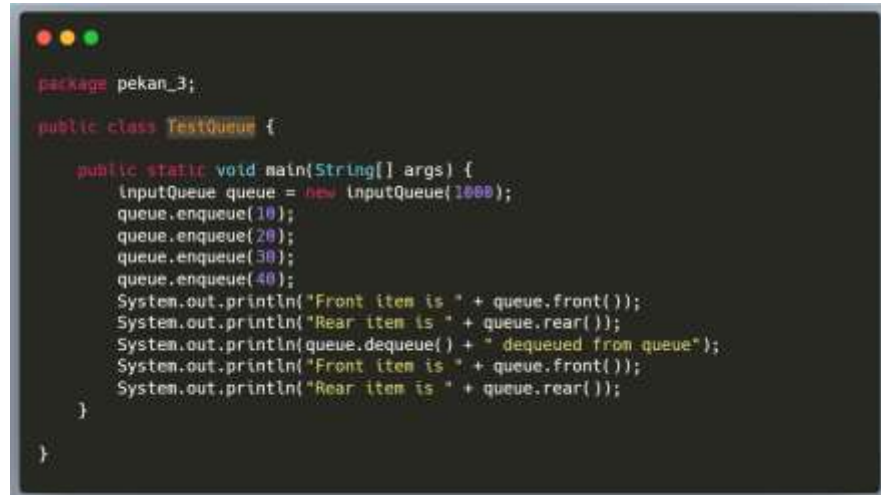
Java tidak menyediakan implementasi default untuk interface `Queue`. Namun, beberapa class bawaan Java seperti `LinkedList` dan `PriorityQueue` dapat digunakan untuk merepresentasikan queue. Masing-masing class ini memiliki kelebihan dan kekurangan dalam hal performa dan urutan elemen yang diambil. Pemilihan implementasi yang tepat tergantung pada kebutuhan spesifik program Anda.

Penggunaan queue sangat luas dalam berbagai skenario pemrograman. Contohnya, queue dapat digunakan untuk memproses tugas dalam urutan kedatangan, seperti pada antrian pencetakan dokumen. Queue juga berguna untuk buffering data, misalnya saat menerima data dari jaringan secara

bertahap. Selain itu, queue berperan penting dalam implementasi algoritma pencarian seperti Breadth-First Search (BFS) pada graf.

III. LANGKAH KERJA PRAKTIKUM

3.1. Membuat dan menguji nilai queue sederhana



```
package pekan_3;

public class TestQueue {

    public static void main(String[] args) {
        InputQueue queue = new InputQueue(1000);
        queue.enqueue(10);
        queue.enqueue(20);
        queue.enqueue(30);
        queue.enqueue(40);
        System.out.println("Front item is " + queue.front());
        System.out.println("Rear item is " + queue.rear());
        System.out.println(queue.dequeue() + " dequeued from queue");
        System.out.println("Front item is " + queue.front());
        System.out.println("Rear item is " + queue.rear());
    }
}
```

Pada baris pertama, kita membuat objek **queue** dengan kapasitas 1000 menggunakan konstruktor **inputQueue(1000)**. Selanjutnya, kita menambahkan empat elemen ke antrian menggunakan method **enqueue()**. Setelah itu, kita menampilkan elemen yang berada di depan dan belakang antrian menggunakan method **front()** dan **rear()**. Kemudian, kita menghapus elemen dari antrian menggunakan method **dequeue()** dan menampilkan elemen tersebut. Terakhir, kita menampilkan kembali elemen yang berada di depan dan belakang antrian setelah melakukan penghapusan.

Dalam program ini, kita dapat melihat bagaimana kelas **inputQueue** digunakan untuk membuat antrian dengan kapasitas yang ditentukan, menambahkan elemen ke antrian, mengambil elemen dari antrian, dan menghapus elemen dari antrian. Selain itu, kita juga dapat melihat bagaimana method **front()**, **rear()**, dan **dequeue()** digunakan untuk mengakses dan mengoperasikan elemen dalam antrian.

Dan output yang dihasilkan:

```
<terminated> TestQueue [Java Application]
10 enqueued to queue
20 enqueued to queue
30 enqueued to queue
40 enqueued to queue
Front item is 10
Rear item is 40
10 dequeued from queue
Front item is 20
Rear item is 40
```

3.2. Membuat class Input yang terpisah

```
package pekan_3;

public class InputQueue {
    int front, rear, size;
    int capacity;
    int array[];

    public InputQueue(int capacity) {
        this.capacity = capacity;
        front = this.size = 0;
        rear = capacity - 1;
        array = new int[this.capacity];
    }

    boolean isFull(InputQueue queue) {
        return (queue.size == queue.capacity);
    }

    boolean isEmpty(InputQueue queue) {
        return (queue.size == 0);
    }

    void enqueue(int item) {
        if (isFull(this)) {
            return;
        }
        this.rear = (this.rear + 1) % this.capacity;
        this.array[this.rear] = item;
        this.size = this.size + 1;
        System.out.println(item + " enqueued to queue");
    }

    int dequeue() {
        if (isEmpty(this)) {
            return Integer.MIN_VALUE;
        }
        int item = this.array[this.front];
        this.front = (this.front + 1) % this.capacity;
        this.size = this.size - 1;
        return item;
    }

    int front() {
        if (isEmpty(this)) {
            return Integer.MIN_VALUE;
        }
        return this.array[this.front];
    }

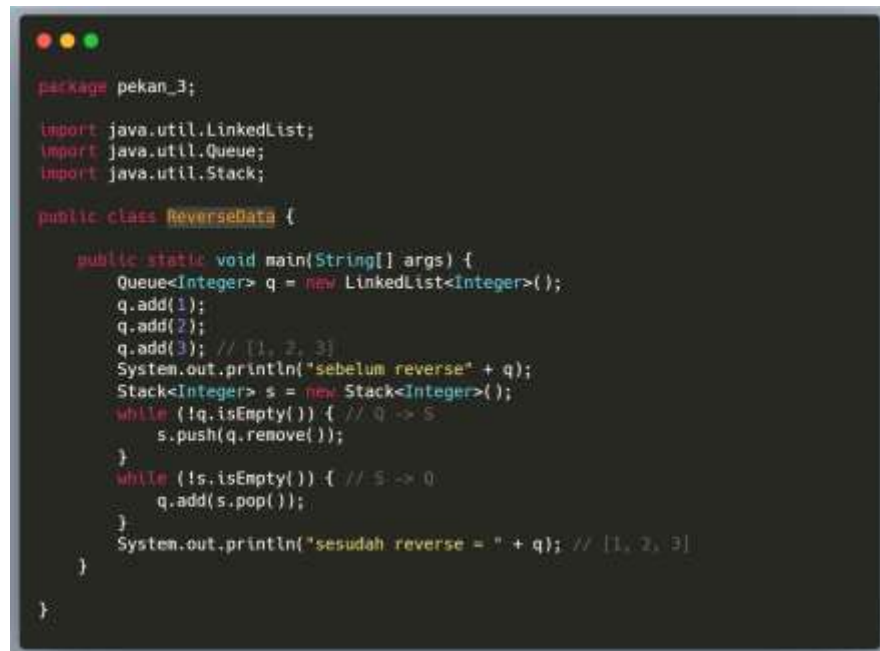
    int rear() {
        if (isEmpty(this)) {
            return Integer.MIN_VALUE;
        }
        return this.array[this.rear];
    }
}
```

- public inputQueue(int capacity):** Konstruktor untuk membuat objek antrian dengan kapasitas yang ditentukan.
- boolean isFull(inputQueue queue):** Method untuk memeriksa apakah antrian penuh atau tidak. Mengembalikan **true** jika antrian penuh dan **false** jika tidak.

- c. **boolean isEmpty(inputQueue queue)**: Method untuk memeriksa apakah antrian kosong atau tidak. Mengembalikan **true** jika antrian kosong dan **false** jika tidak.
- d. **void enqueue(int item)**: Method untuk menambahkan elemen baru ke antrian. Jika antrian sudah penuh, maka tidak akan menambahkan elemen baru.
- e. **int dequeue()**: Method untuk menghapus elemen dari antrian. Jika antrian kosong, maka akan mengembalikan nilai **Integer.MIN_VALUE**.
- f. **int front()**: Method untuk mengambil elemen yang berada di depan antrian tanpa menghapusnya. Jika antrian kosong, maka akan mengembalikan nilai **Integer.MIN_VALUE**.
- g. **int rear()**: Method untuk mengambil elemen yang berada di belakang antrian tanpa menghapusnya. Jika antrian kosong, maka akan mengembalikan nilai **Integer.MIN_VALUE**.

Dalam program ini, Queue diimplementasikan menggunakan array circular, yang memungkinkan penambahan dan penghapusan elemen dengan efisiensi waktu yang konstan. Variabel **front** digunakan untuk mencatat indeks elemen pertama di queue, sedangkan variabel **rear** digunakan untuk mencatat indeks elemen terakhir di queue. Variabel **size** digunakan untuk mencatat jumlah elemen yang ada di queue.

3.3. Melakukan reverse data



```

package pekan_3;

import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

public class ReverseData {

    public static void main(String[] args) {
        Queue<Integer> q = new LinkedList<Integer>();
        q.add(1);
        q.add(2);
        q.add(3); // [1, 2, 3]
        System.out.println("sebelum reverse" + q);
        Stack<Integer> s = new Stack<Integer>();
        while (!q.isEmpty()) { // Q -> S
            s.push(q.remove());
        }
        while (!s.isEmpty()) { // S -> Q
            q.add(s.pop());
        }
        System.out.println("sesudah reverse = " + q); // [1, 2, 3]
    }
}

```

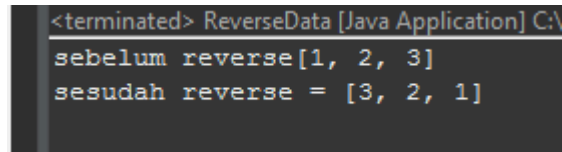
Pada baris pertama, kita menggunakan konstruktor LinkedList untuk membuat objek q dengan tipe data Queue(). Kemudian, kita menggunakan metode add untuk menambahkan tiga elemen ke antrian. Selanjutnya, kita menggunakan metode toString untuk menampilkan isi antrian sebelum reverse.

Selanjutnya, kita menggunakan konstruktor Stack() untuk membuat objek s dengan tipe data Stack. Kemudian, kita menggunakan metode isEmpty untuk melakukan looping selama antrian q tidak kosong. Pada setiap iterasi, kita menggunakan metode remove untuk menghapus elemen terdepan dari antrian q dan menggunakan metode push untuk menambahkan elemen tersebut ke atas stack s.

Selanjutnya, kita menggunakan method isEmpty() untuk melakukan looping selama stack s tidak kosong. Pada setiap iterasi, kita menggunakan method pop() untuk mengambil elemen teratas dari stack s dan menggunakan method add untuk menambahkan elemen tersebut ke belakang antrian q.

Terakhir, kita menampilkan isi antrian setelah reverse menggunakan method **toString()**. Dari hasil program ini, kita dapat melihat bahwa isi antrian telah berhasil diurutkan secara terbalik menjadi **[3, 2, 1]**.

Dan output yang dihasilkan:



```
<terminated> ReverseData [Java Application] C:\
sebelum reverse [1, 2, 3]
sesudah reverse = [3, 2, 1]
```

IV. KESIMPULAN

Dengan menggunakan array dan daftar yang terhubung, program pertama dan kedua menjalankan antrian (queue), yang merupakan struktur data yang mengikuti prinsip "first-in, first-out" (FIFO), yang berarti elemen yang masuk pertama akan keluar pertama. Program pertama menentukan kapasitas antrian dan melakukan beberapa operasi dasar seperti enqueue (menambahkan elemen ke antrian), dequeue (mengambil elemen dari depan antrian), front (menampilkan elemen pertama di antrian), dan enqueue (menambahkan elemen ke antrian).

Program ketiga menggunakan LinkedList dan Stack untuk menambahkan beberapa angka ke antrian dan kemudian mengurutkannya secara terbalik menggunakan stack. Ini dilakukan dengan mengambil semua elemen dari antrian dan menumpukkannya ke stack, kemudian mengambil semua elemen dari stack dan menambahkannya kembali ke antrian. Oleh karena itu, urutan elemen dalam baris akan terbalik. Kita dapat mengatakan dari ketiga program ini bahwa antrian adalah struktur data yang penting dan bermanfaat dalam berbagai situasi, terutama dalam pengelolaan data yang membutuhkan urutan tertentu. Antrian dapat digunakan untuk mengatur urutan eksekusi berbagai proses atau thread atau untuk menyimpan data yang akan diproses nanti. Selain itu, algoritme tertentu, seperti algoritme pencarian lebar pertama, juga dapat bekerja dengan antrian.